No-wait Idea with SRMs

Eliminate Update task delay

Thu, Apr 5, 2001

In the Linac control system, each front-end is connected via arcnet to one or more SRMs. At the start of every cycle, each front-end sends a request message, usually broadcast, to each SRM. The meaning of the request is "read all your data and send it back in a single arcnet frame." Each SRM knows what hardware is connected to it according to entries in its "data access table" housed in its nonvolatile memory. It takes some time to do this, typically about 10–15 ms, depending on the number of SRMs and how much data each returns. During this time, the front-end cannot really do anything. It has started its Update task, and its main job is to update the data pool "all at once." This means that to an outside user, the data pool should be seen as being updated at one time in the cycle; it should not be possible to observe a partially updated data pool. This rule is maintained to achieve the objective of correlated data. Any data delivered during one 15 Hz cycle was all measured during that cycle. This note suggests an idea for avoiding the Update task delay while it awaits SRM replies.

The Data Access Table in the front-ends—not to be confused with its namesake in the SRMs—consists of a list of "instructions" that specify the steps to be followed to update the data pool. Often, the first instruction says "send out the SRM request message" as described above. If this is really the first instruction, nothing has yet been updated in the data pool. The basis of the idea is to arrange that the front-ends start their cycle a little later. At the usual 3 ms past the reset clock event, which is expected to be the 0C event in the new PowerPC-based Linac controls, we would send out the SRM data request. Then we could have the Update task block for a time that approximates when the SRM data is likely to be ready. When it again proceeds, it should not have to wait so much for SRM replies, if at all.

A possible objection to this idea is that a data request that arrives during this timeout period may likely be honored, and immediate reply data will be returned. This data will include data measured on the previous cycle. But this would happen if the request arrived just before the start of the cycle, too. It is important to remember that all front-ends in a project (such as Linac) operate at the same time; every one is triggered by the same clock event plus delay. If one node has no SRMs, say, it can deliver data quite early, even during the time that another node is timing out before it calls for the SRM reply data. Such data could be received, but it could not be put to good use. According to this scheme, the Update task has suspended its activity for a little while. It is not in a position to be doing its other tasks until it finally finishes updating the data pool. A page application cannot run until it has its turn, which is after every other periodic task has run.

Let us ask the question, what can be done in terms of useful work during the time that the Update task is timing out? Many jobs cannot be done. The local applications cannot run, because they often need fresh data to do their job. The Alarm task cannot run, because it needs fresh data in the data pool to examine. The Console task may be able to run, but to what end? The page application, as mentioned above, is the "last hog to the trough." Considering each task , one by one, there is nothing of significance that can take advantage of the new free time available to the CPU. The conclusion, therefore, is that there is no reason not to have the Update task silently await the arrival of fresh SRM data.